B58: MIPS Tetris - Checkpoint Demo Design Document
Zohair Syed

Overview - 256 x 256 pixels, 8 units; works on Mars and Saturn

Memory Layout

- Stores the display framebuffer/keyboard
- Stores the board (keeps track of each cell in the grid and whether it has a block committed to it or not - holds 0/1 booleans)
- Stores a list of 'active' states
    - Current row/column, column, orientation of falling piece
    - Current rotation block
- Stores tetrominos

Routines/Main Flow

The program starts by calling **main**. Main initializes the framebuffer, keyboard and the list of active states above. It then moves forward to **init_game,** which initializes and draws the static UI (the borders and checkerboard pattern). At this stage, a current piece is chosen (currently hardcoded) to be the start.

Next, the program goes into the main **game_loop,** which polls for the user's input and breaks into a switch statement to decide the next course of action. If the user presses A, S or D we go into **move_left**, **move_down** or **move_right** (if the user presses Q, the program exits)**.** These labels then load arguments for moving the piece accordingly and then call **can_move**.

Can_move iterates over the pieces, and performs collision detection to see if the piece can move. If the piece can move, the function returns with 1 and we move on to the next stage. First, **clear_loop** clears the current piece and then **start_drawing** draws the updated coordinates. Finally, **lock_piece** commits the new coordinates to *board* for future detection. After this step (or if the piece was determined to not be movable), the function jumps back to **spawn_piece** to generate a new (currently hardcoded) piece and continue the process.

THe final major part of this checkpoint is rotation. When W is pressed, the **try_rotate** label is called, which loads many active states. 4 copies of each piece are located in memory, so we need to track which one to apply on the rotation. This is stored in *active_orientation*. After determining whether a piece is safe to rotate in **can_rotate**, the current piece is cleared and the active piece pointer points to the rotated piece in memory. This cycle continues.

Structures

- Currently, tetrominos are stored as 4x4 grids consisting of 16 elements (4 bytes)
- 4 copies of each tetronimo exist for each rotation (90, 180, 270 degrees)
- Colours and addresses are currently stored as bytes/words in memory
- Active states are stored as words in memory
- X_piece_rotations is a struct with 4 elements (all words)
    - Each word is a pointer to one of X's rotated tetronimo pieces